DevOps community
Build you own CI

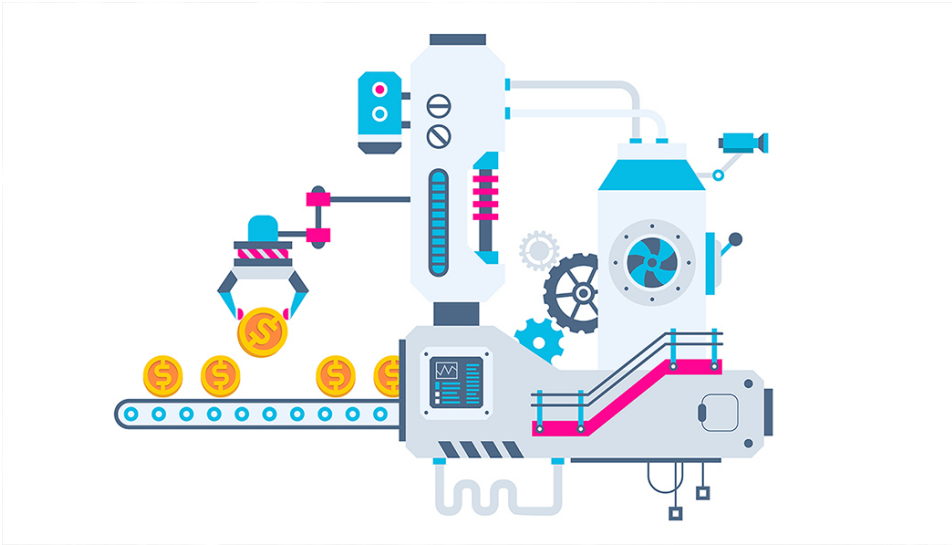
October 2018 – Cyril Marin & René Ribaud

CGI

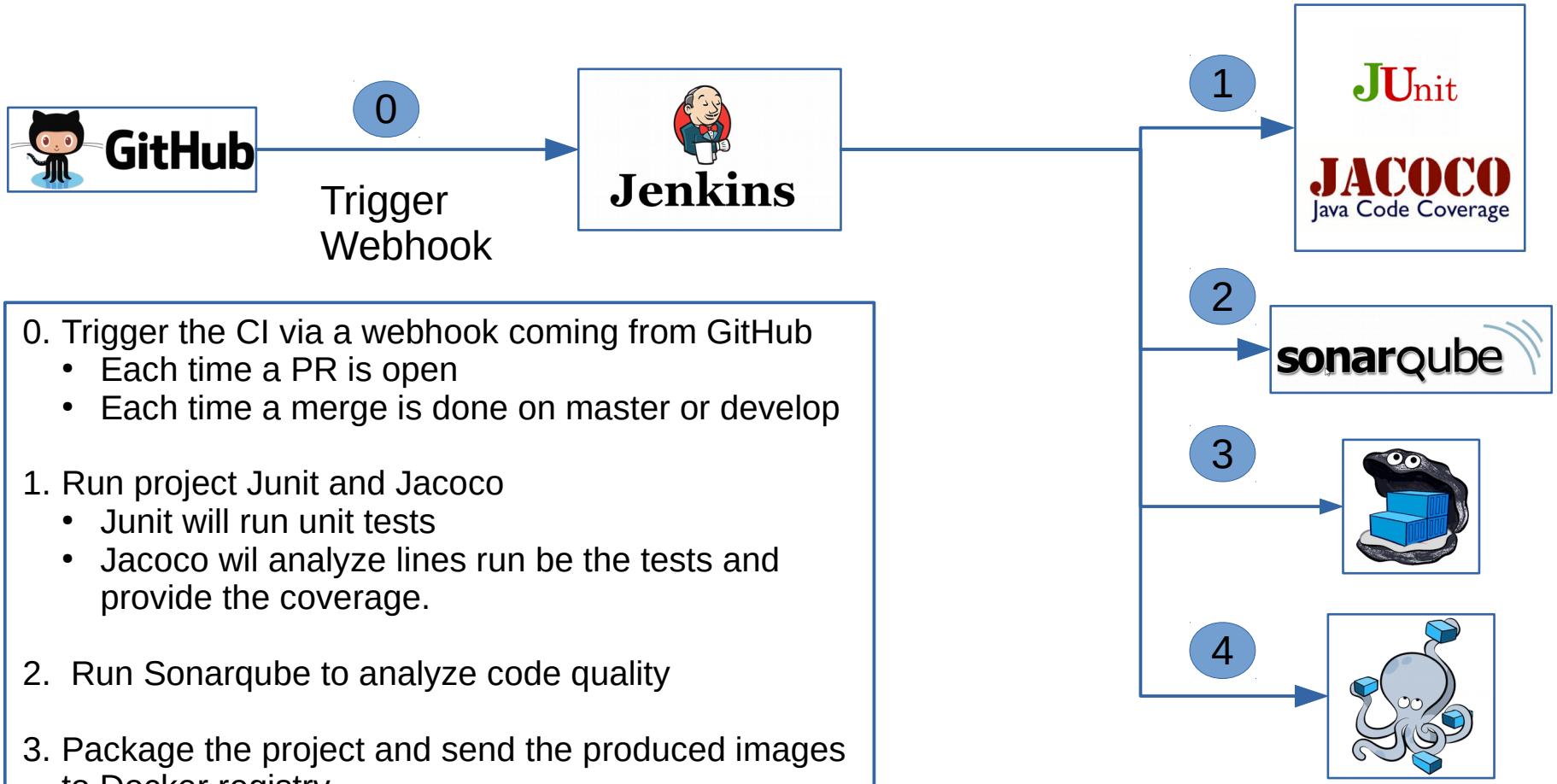
Experience the commitment®

Goal of the session

- Build a simple Continuous Integration for a “hello world project”.
  - Capitalize on our Docker knowledge, tools will be installed using Docker images
  - Infrastructure used:
    - Project code will be hosted on Github
    - 1 x VM will host CI tools
    - 1 x VM will be used to deploy the project.



CI/CD Pipeline



0. Trigger the CI via a webhook coming from GitHub
   - Each time a PR is open
   - Each time a merge is done on master or develop

1. Run project Junit and Jacoco
   - Junit will run unit tests
   - Jacoco wil analyze lines run be the tests and provide the coverage.

2. Run Sonarqube to analyze code quality

3. Package the project and send the produced images to Docker registry

4. Trigger deployment on our node using Docker compose

CGI

Deploy 2 x VM

- Use the portal at registry.uggla.fr to create 2 Vms
  - Use docker 18.04 images

- In the next part of the document we will refer to the VM as
  - vmci → VM that will contains tools, warning this VM should be created with 4GB of memory.

  - vmdev → VM that will be used to deploy the project

CGI

THE project !

Simple HelloWorld project

- URL : https://github.com/uggla/HelloWorld

- Fork the project using Github button
  To create your own project

- Clone your own project on the vmdev
  - git clone <your project url)

- Build and run the fantastic project using maven docker image
  (note maven images contains openjdk)
  - docker run --rm -it -v /tmp/.m2:/root/.m2 -v $(pwd):/usr/src/myproject -w /usr/src/myproject maven:3-jdk-8-slim mvn package

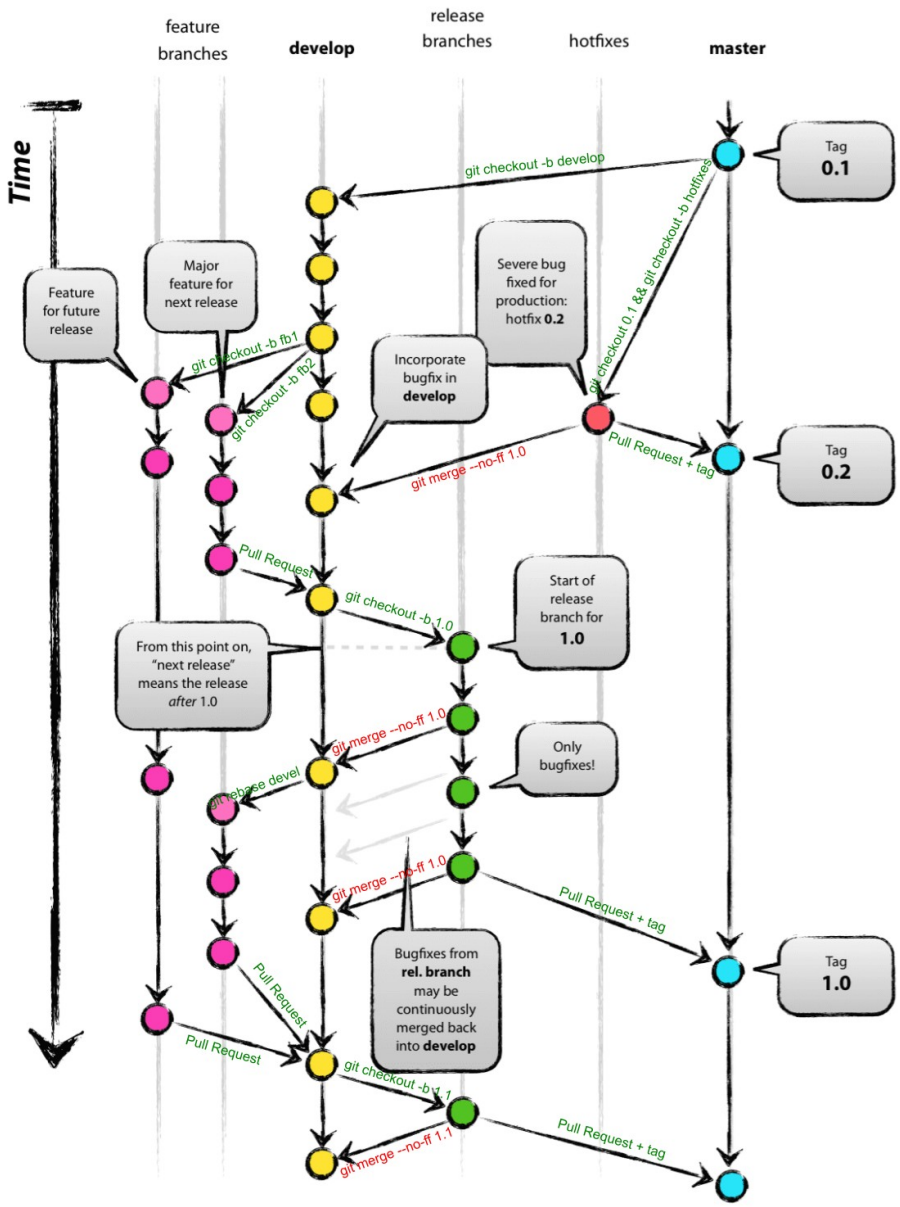
  - docker run --rm -it -v /tmp/.m2:/root/.m2 -v $(pwd):/usr/src/myproject -w /usr/src/myproject maven:3-jdk-8-slim java -jar HelloWorld-0.1.0.jar

CGI

Git branching model

This project will be managed using Git branching model, explaining this model in depth is not part of this session.

- Create the required develop branch
  - git checkout -b develop



CGI

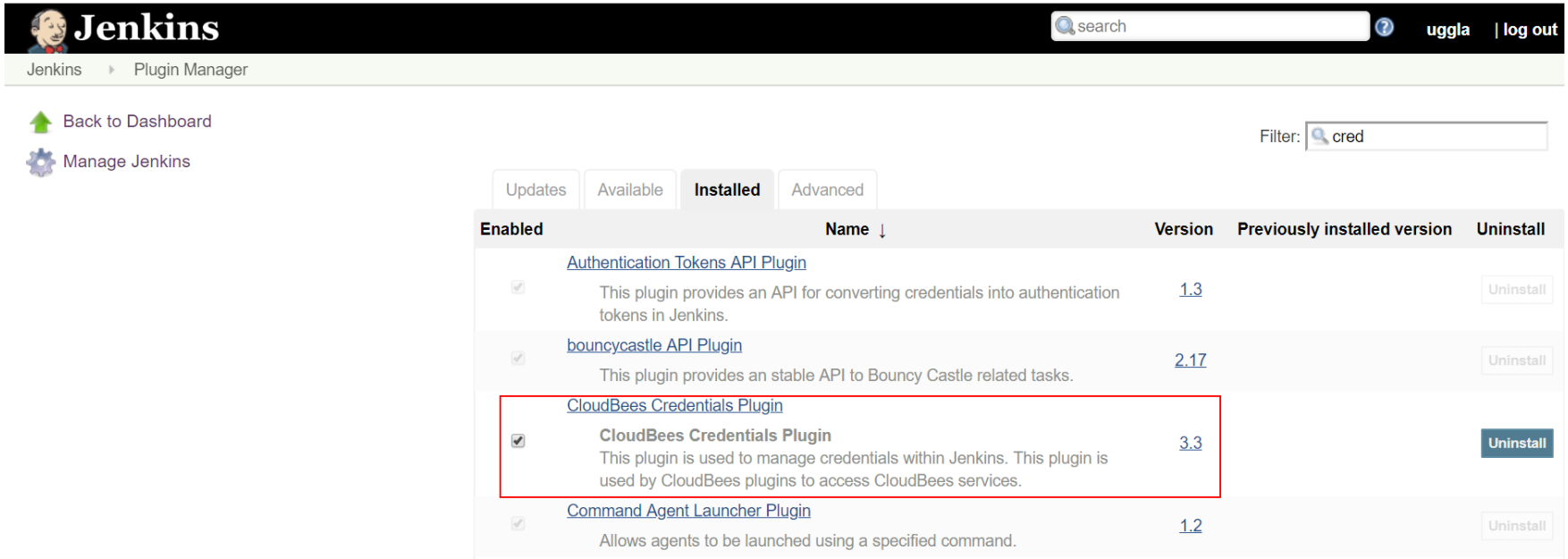
Install the tools

- Install Jenkins
- Install Sonar

- To do it:
  - Go into project ci directory
  - Run run.sh
    - Create appropriate user for Jenkins
    - Set correct permission on the Docker socket
    - Install registry certificate from registry.uggla.fr

Configuration Jenkins #1

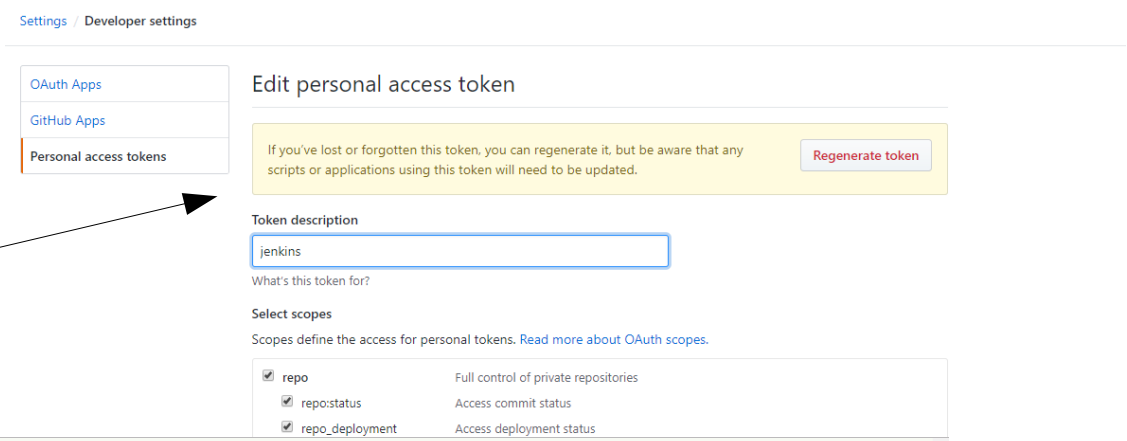
- Go to http://<vmci ip>:8080
- Go ahead with the setup by installing recommended plugins
- Create an admin user, please set a strong password as the vm is exposed to the internet
- Install Cloubees credentials plugins + blue oceancd

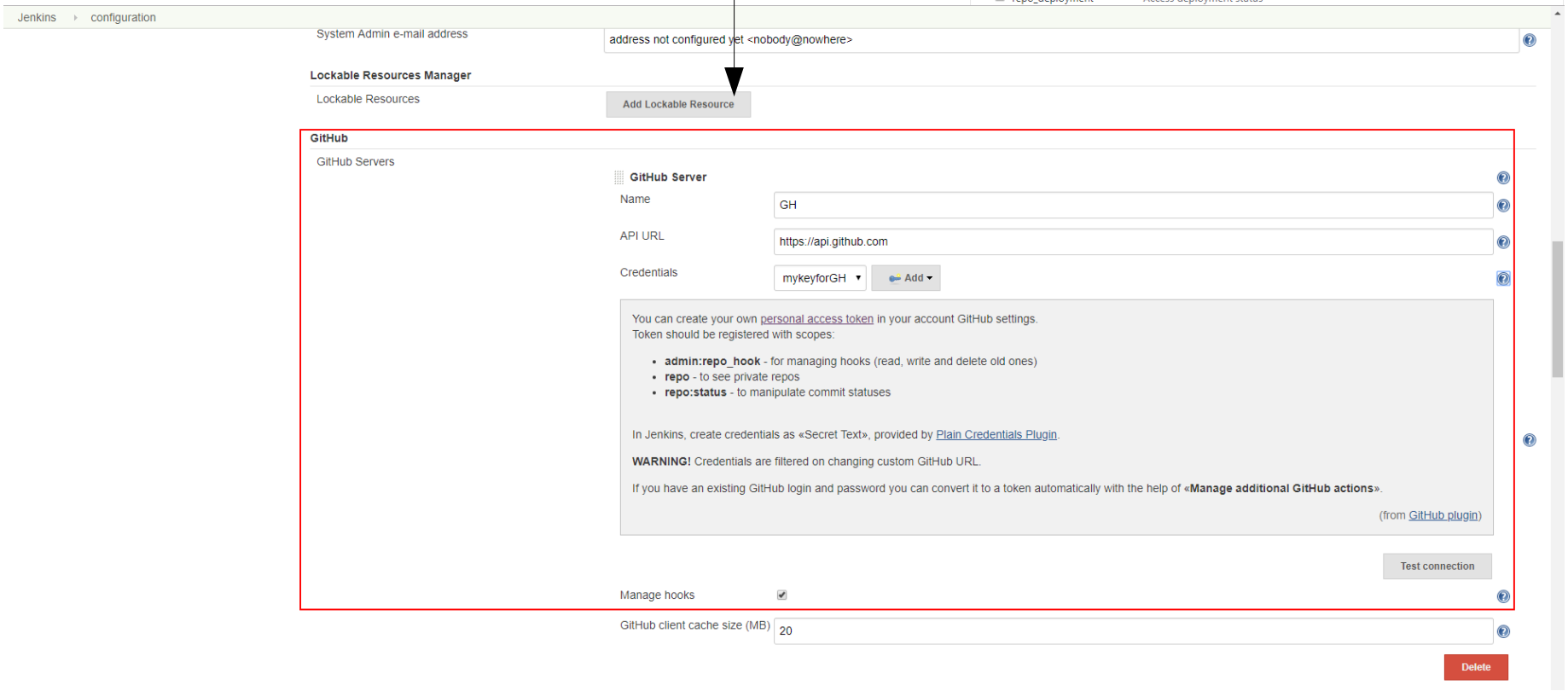


CGI

Configuration Jenkins #2

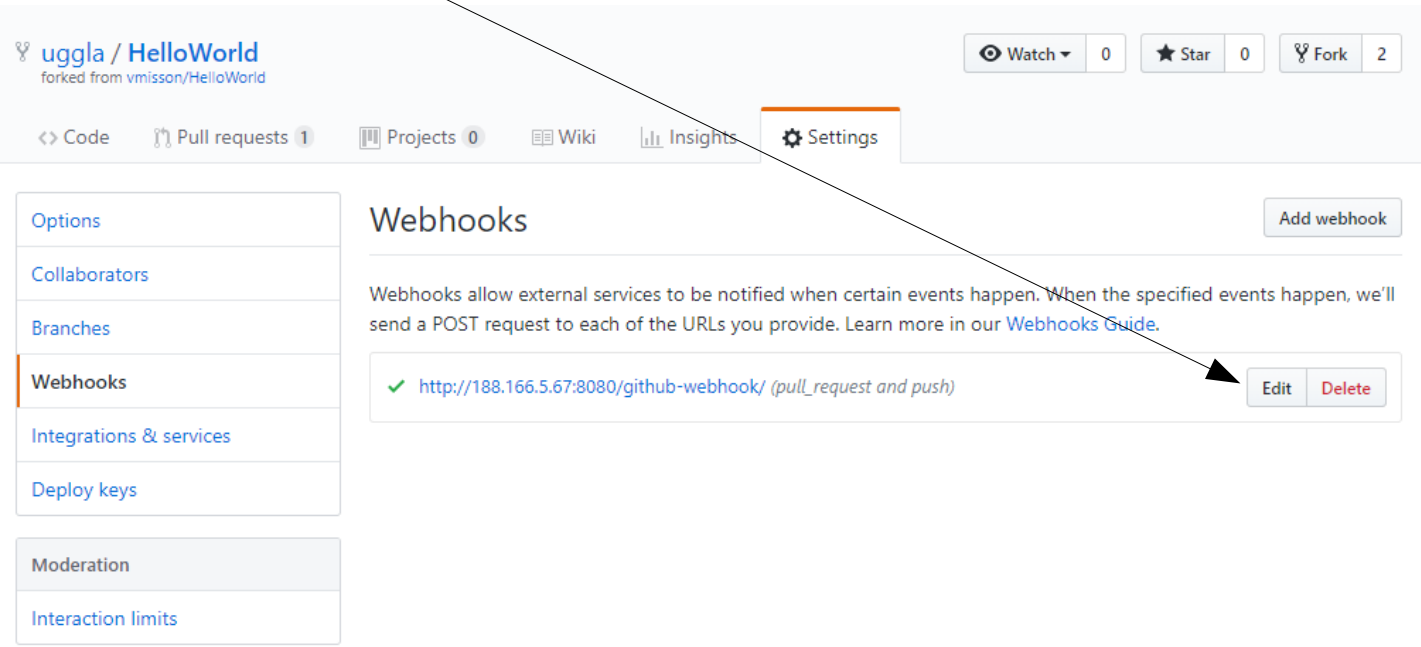
- Configure the trigger
  - Create a personal access token to your Github account
  - Configure a Github server within Jenkins





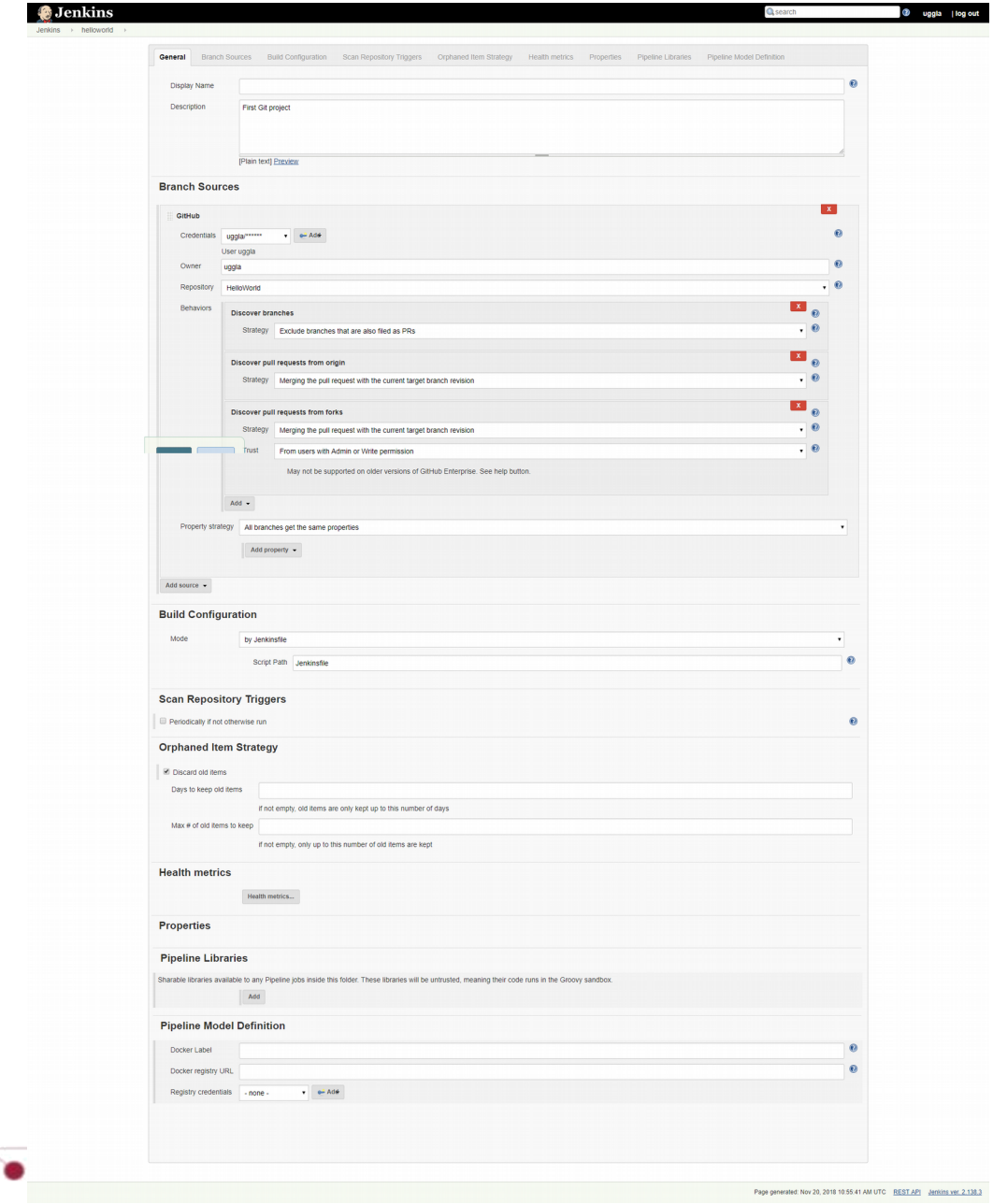
Configuration Jenkins #3

- Configure the trigger
  - Check the Github repository webhook configuration
  - Click the edit button in order to have details and checks



Configuration Jenkins #4

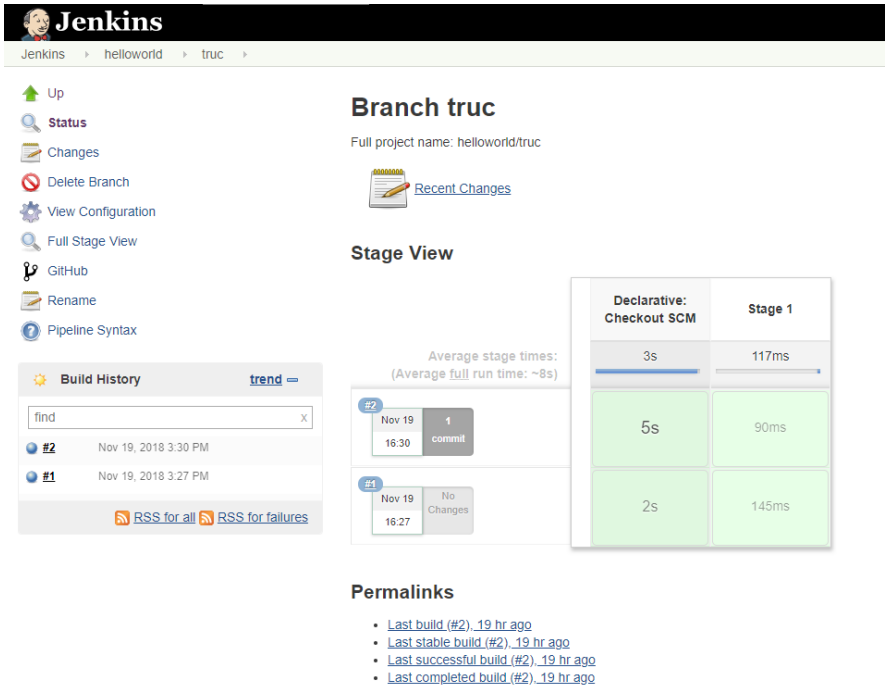
- Create a multibranch pipeline
  - Set the correct Github credentials
  - Set the correct repository
  - Set options as default settings



Define the pipeline #1

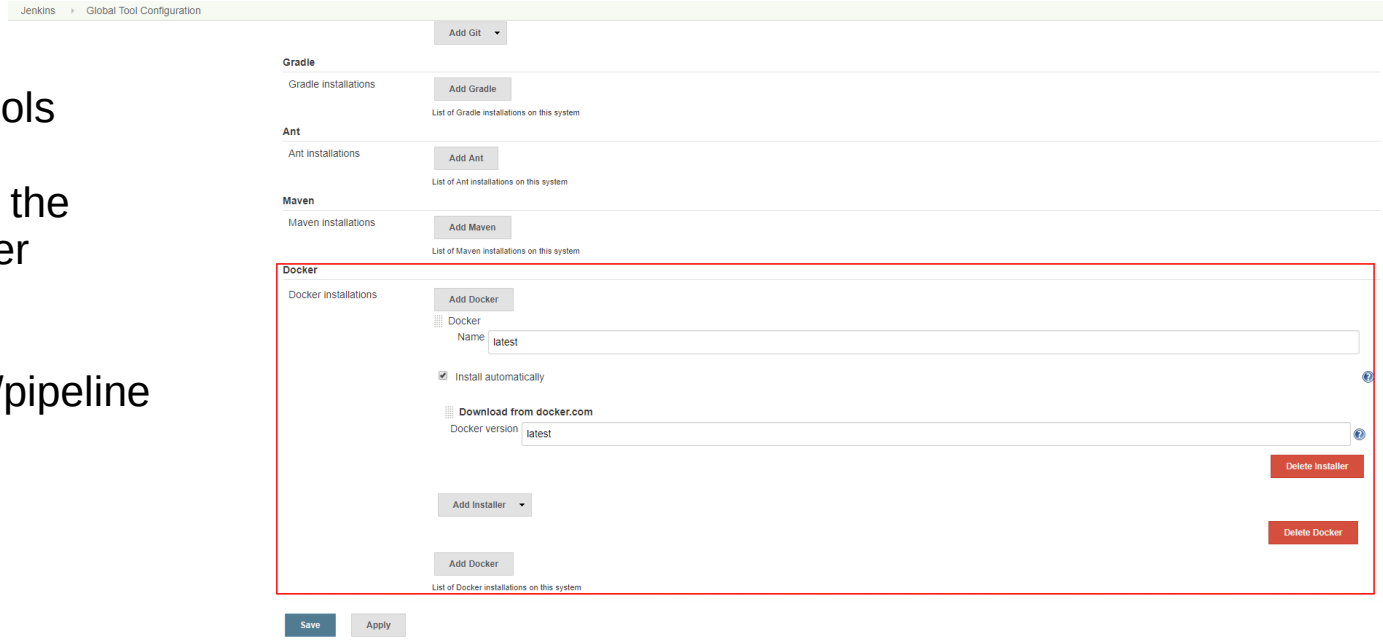
- Check CI
  - Create a small Jenkinsfile at the root of the project
  - Commit and push the file
  - A job should be successfully run and logs must contain the echos from Jenkinsfile
  - If it works, create a new branch and an associated PR.

```
pipeline {
    agent any

    stages {
        stage('Stage 1') {
            steps {
                echo 'Hello world!'
                echo 'Coucou !'
                echo 'Welcome !'
            }
        }
    }
}
```



Define the pipeline #2

- Build the project
  - Add Docker to the list of tools
  - Update Jenkinsfile to build the project using maven Docker container
    Doc : https://jenkins.io/doc/book/pipeline/docker/




```
pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                echo 'Hello world!'
                echo 'Coucou !'
                echo 'Welcome !'
                script {
                  docker.withTool('latest') {
                    docker.image('maven:3-jdk-8-slim').inside("-v $WORKSPACE:/usr/src/myproject:rw -v
$HOME/.m2:/root/.m2:rw -w /usr/src/myproject") { c ->
                      sh "mvn clean package"
                    }
                  }
                }
            }
        }
    }
}
```

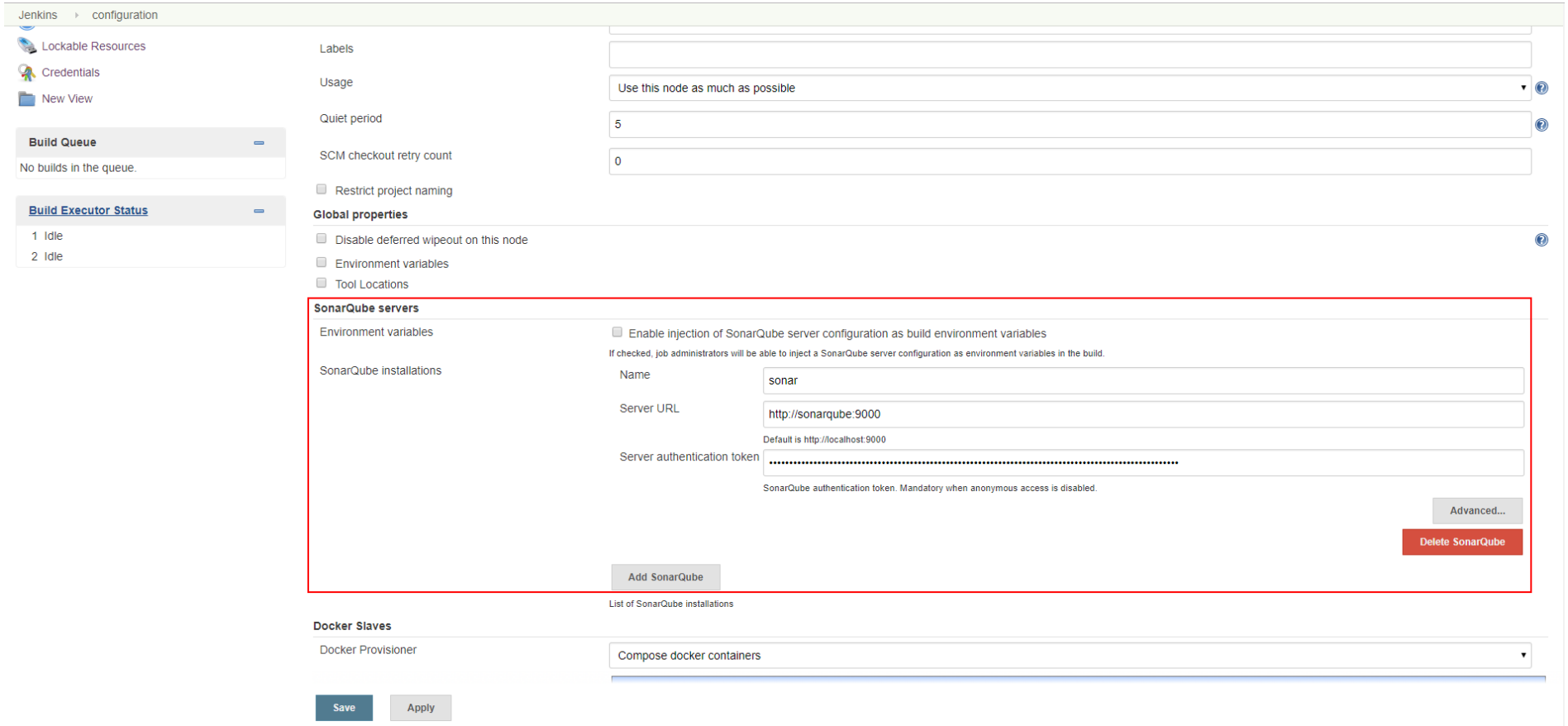
Add Sonar quality check #1

- Configure sonar
  - Connect to sonar http://<vmci ip>:9000
  - Login with admin:admin and set a token for your project (keep the token somewhere it will be used later)
  - Set a strong password as the vm is internet facing
  - Install sonarqube scanner for Jenkins plugin

CGI

Add Sonar quality check #2

- Configure sonar
  - Configure the plugin



Add Sonar quality check #3

- Configure sonar
  - Update pipeline allowing sonar analysis

```
pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                echo 'Hello world!'
                echo 'Coucou !'
                echo 'Welcome !'
                script {
                  docker.withTool('latest') {
                    docker.image('maven:3-jdk-8-slim').inside("-v $WORKSPACE:/usr/src/myproject:rw -v
$HOME/.m2:/root/.m2:rw -w /usr/src/myproject") { c ->
                      sh "mvn clean package"
                    }
                  }
                }
            }
        }
        stage('Quality check') {
            steps {
                script {
                  docker.withTool('latest') {
                    docker.image('maven:3-jdk-8-slim').inside("--network ci_cinet -v $WORKSPACE:/usr/src/myproject:rw -v
$HOME/.m2:/root/.m2:rw -w /usr/src/myproject") { c ->
                      withSonarQubeEnv('sonar') {
                        sh 'mvn sonar:sonar'
                    }
                  }
                }
              }
            }
        }
    }
}
```

Package the application #1

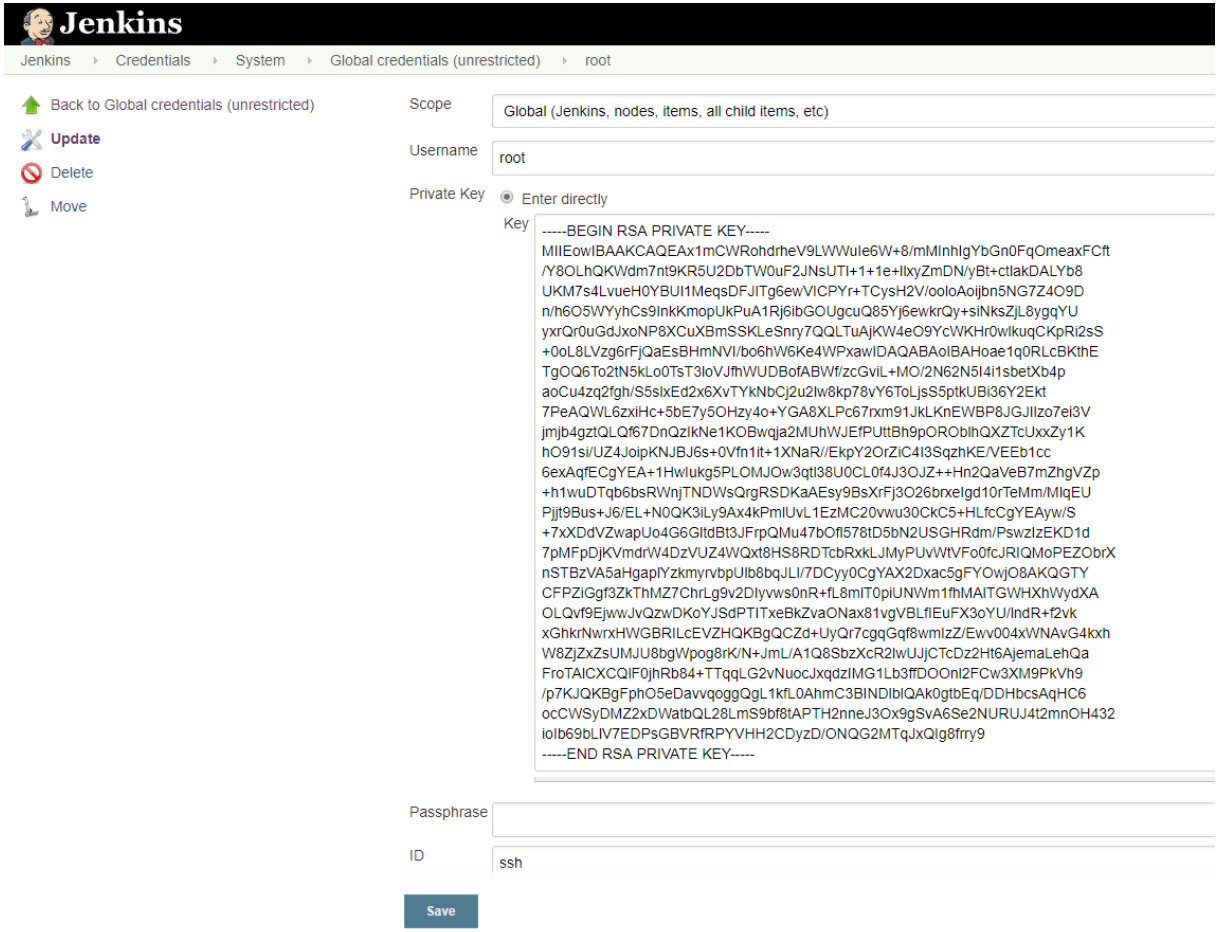
- Add a stage to package the application
  - Update pipeline allowing packaging

```
def IMAGENAME='named-your-image'

pipeline {
    agent any

    stages {
        stage('Build') {
            steps {
                echo 'Hello world!'
                echo 'Coucou !'
                echo 'Welcome !'
                script {
                  docker.withTool('latest') {
                    docker.image('maven:3-jdk-8-slim').inside("-v $WORKSPACE:/usr/src/myproject:rw -v $HOME/.m2:/root/.m2:rw -w
/usr/src/myproject") { c ->
                     sh "mvn clean package"
                    }
                  }
                }
            }
        }
        stage('Quality check') {
            steps {
                script {
                  docker.withTool('latest') {
                    docker.image('maven:3-jdk-8-slim').inside("--network ci_cinet -v $WORKSPACE:/usr/src/myproject:rw -v
$HOME/.m2:/root/.m2:rw -w /usr/src/myproject") { c ->
                     withSonarQubeEnv('sonar') {
                       sh 'mvn sonar:sonar'
                    }
                  }
                }
              }
            }
        }
        stage('Package app') {
            steps {
                script {
                  docker.withTool('latest') {
                    docker.withRegistry('https://registry.uggla.fr/') {
                      def myImg = docker.build("${IMAGENAME}:${env.BUILD_ID}")
                      myImg.push()
                    }
                  }
                }
            }
        }
    }
}
```

Deploy the application #1

- Deploy and run the application
  - Add the Docker registry certificate to vmdev
  - Create a ssh root keys
  - Load private keys to into Jenkins credentials

```
REGISTRY_FQDN=registry.uggla.fr

# Get certificate from registry
curl -L http://${REGISTRY_FQDN}:81/ca.crt >
/usr/local/share/ca-certificates/ca-registry.crt
update-ca-certificates
systemctl restart docker.service
```



Deploy the application #2

- Deploy and run the application
  - Add Jenkins ssh pipeline steps plugin
  - Update Jenkinsfile to add deployment

```
def IMAGENAME='named-your-image'
def VMDEV='188.166.48.108'

pipeline {
    agent any

    ...
        stage('Deploy app') {
            steps {
                script {
                  withCredentials([sshUserPrivateKey(
                  credentialsId:'ssh',
                  keyFileVariable:'pkey',
                  PassphraseVariable:'',
                  usernameVariable:'user')]){
                    sh "ssh -i ${pkey} -o StrictHostKeyChecking=no root@${VMDEV} docker run
-i registry.uggla.fr/${IMAGENAME}:${env.BUILD_ID}"
                  }
                }
            }
        }
    }
}
```